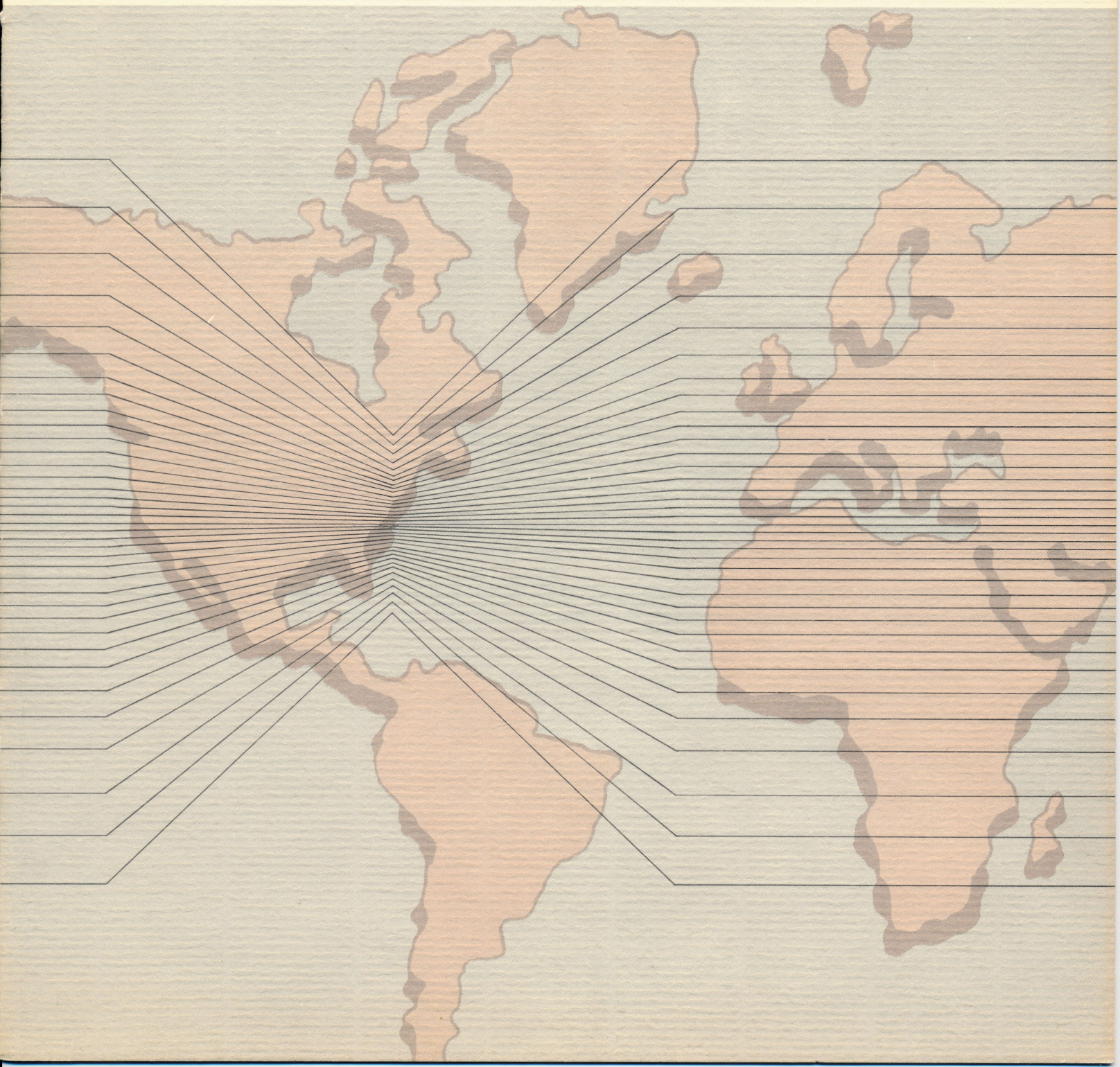
I. P. Sharp Associates

SHARP APL File System



EASY ACCESS TO BULK DATA

via SHARP APL Files

High-performance high-security access to data files is built into the SHARP APL system.

Using APL files, you can store as much (or as little) data as you like. You can also make use of very extensive public data files regularly maintained and updated by Sharp.

Why files?

In an APL system, all calculations take place in a **workspace**. While a workspace is quite large (100,000 bytes at present), its size is fixed -- and finite. Data for a small or medium job can usually be stored within a workspace. But if your job requires something larger, you'll need files. A large-scale job calls for two things:

(a) breaking the work into steps that deal with just a part of the data at any one time;

(b) keeping the data that's not required for the current step somewhere other than in the workspace.

A file gives you a way to store almost any amount of data outside the workspace. Storage is divided into **components,** each of which is an APL data array that can fit into the workspace.

Why a "built-in" system?

The SHARP APL file system itself was built especially for work with APL. It's designed for the storage of APL data, and tuned to work with a high-performance time-sharing system.

SHARP APL provides, as part of the APL language, primitive (that is, built-in) functions for transferring data between the workspace and a file. There are also functions for creating a new file, deleting a file, and establishing what uses -- if any -- you'll permit other people to make of files that you own.

Because the file system is especially built for use with APL, you get fast response and a minimum of the esoteric protocols and controls needed to establish files in other systems.

Because the file access is by APL primitives, you get immediate access to the carefully constructed code that lies behind the APL system, without any need for subroutines that might otherwise require space to store and time to run.

How you use files

Many of the APL functions that use a file use three numbers to refer to a particular component of a particular file. For example, the expression

```
X←□READ 15  186  8223
```

means that you want to create in the workspace an array called *X*, and that *X* is to get its value by reading from the file you've designated as number 15, at component number 186. The third number is a pass-number; if the file owner has set things up this way, you have to include a third number, and it must match the value specified by the owner. It's a security check. If the pass-number has not been enforced, you can leave it off your access requests.

To establish which file you're talking about, before using it, you have to **tie** it. You provide the name (including, if necessary, the account number of its owner), and state the number by which you wish to refer to it. Provided the file exists, is available, and you are authorized to use it, your request is accepted. From then on throughout the session you use the tie number as part of the expression that reads or writes from a file. The example above shows how to read component 186 of file 15. Here are two other examples:

```
Y □REPLACE 15  187
```

replaces the current value of component 187 of file 15 with the array which in the workspace is called *Y*. If no pass-number has been set, the third number can be left out; otherwise, you'd have to provide a correct pass-number as well.

```
Z □APPEND 15
```

means that the array called *Z* in the workspace is to be written as the next record after the last record thus far used in file 15. For instance, if file 15 currently contained 200 components, *Z* would be stored as the 201st.

Data integrity

SHARP APL files are backed up daily to safeguard against damage, or, what is more common, against user errors that result in a user mistakenly losing some data. The SHARP APL File Subsystem manual describes procedures to minimize difficulties in the event of a system failure in the middle of a job using files.

Data security

A file belongs to a user. Like a workspace, the full name of a file consists of the owner's account number and a name.

For each file, there is an **access matrix**. This list created by the user shows which people are authorized to use the file, what kinds of access they can have, and what pass-number they must provide to access it. If the owner doesn't explicitly set the access matrix, then **no one** else can make any use of a file.

Another APL function lets the owner of a file list the names of his or her own files, and on request, list the names of files to which another user has expressly granted him access.

Quick summary of features in the SHARP APL file system

- **File primitives**. Files are created, read, and written using APL primitives.

- **Random access**. All reading and writing is indexed. That is, you refer to the component by mentioning its component number (like "Give me that 156th component").

- **APL self-describing data**. Each component of a file is an APL array. That means that stored with it in the file is the same data description it has in the workspace, showing its size, shape, type, etc. (but not the name by which it was known in the workspace).

- **Variable length, variable type**. The various components of a file may differ in size, shape, type. In SHARP APL, you don't declare in advance what kind of data will be written into a file or what size each file component will be. The system simply accepts whatever data you happen to send. Each component is independent in size, shape, and content from all other components of the same file.

- **Expansible components**. A component of a file can change in size, shape, and type. You can write a component so that it contains a data array larger or smaller than the one it contained previously.

- **Components of any size**. A component of a file can be an array of any size and shape, although workspace size provides an implicit limitation.

- **Program overlays in files**. A file component must be a single data array. But, using the special data type called **package**, a single array can contain definitions of many programs, some or all of which are then readily transferred to the workspace when needed.

- **Shared files**. The owner of a file can permit others to use it **simultaneously**. There's provision to **hold** a file to prevent errors when one of the users makes an update at the same time that someone else is using the file.

- **Back-up protection against data loss**. SHARP APL files, like workspaces, are backed up daily. This allows simple recovery if data is inadvertently destroyed. The system operators keep on a daily basis an archive of each file component that has been newly written since the day before. So when you or your program erase data you really wanted, you can arrange to have it retrieved within a matter of hours.

- **Components individually timestamped**. Each file component contains within it a "timestamp" showing who wrote it and when. (This is what permits the back-up services to identify components that have been changed.)

- **File owner gets storage charges**. Each file has an owner, usually the person who created it. The owner is charged for storing the file, but only for the records actually written in it.

- **User-set limits control growth**. At creation, the owner of a file sets a **file reservation** -- the maximum size to which the file may grow. If the file turns out to have too restrictive a limit, the owner can reset it any time. The system managers set (and can reset) a maximum for the sum of all reservations a user can make.

- **Simultaneous use of many files**. One user can access several files at the same time. A maximum of 20 files can be tied by a user at a time.

- **Access control for security**. Each file has an **access matrix** showing who may use it, and for what uses. In that way, the owner of a file can designate what other people, if any, can have access to the file, and what kind of access they can have. The owner can also set **pass-numbers** that other users must supply with each command to make use of your file.

- **Conversion for external files**. Data in non-APL files can be converted by a background job that can be run on request. The conversion consists of generating for each component a data description, specifying the shape and type of the data; the same description must apply to the entire array. Similarly, on request an APL file can be converted to a form used in non-APL installations.

Printed in Canada
February 1978